Mesh motion alternatives

CFD with OpenFOAM

Andreu Oliver González

14/12/2009

Outline

- Introduction
- Mesh motion approaches and classes
- Procedure to define a mesh with motion
- Explanation of *dynamicInkJetFvMesh* class
- Modification of *dynamicInkJetFvMesh* class
- Conclusions

Introduction

- Overview of the different classes to define a mesh with motion.
- Deep description of the *dynamicInkJetFvMesh.*
- *icoDyMFoam* and *turbDyMFoam* are the solvers for problems with moving mesh in version 1.5.x.
- In 1.6.x, pimpleDyMFoam collects both solvers.

Mesh motion approaches and classes

- Automatic mesh motion (*dynamicFvMesh*)
- Topological mesh changes (*topoChangerFvMesh*)

Mesh motion approaches and classes

- Automatic mesh motion (*dynamicFvMesh*):
  - *staticFvMesh* – no motion.
  - *dynamicMotionSolverFvMesh* – relatively small changes.
  - *dynamicInkJetFvMesh* – similar to the on before.
  - *dynamicRefineFvMesh* – refinement or unrefinement of the mesh.
  - *solidBodyMotionFvMesh* – to describe solid body motion.

Mesh motion approaches and classes

- Topological mesh changes (*topoChangerFvMesh*):
  - *linearValveFvMesh* – to use sliding meshes at the interface of 2 pieces in relative linear motion.
  - *linearValveLayersFvMesh* – as the one before but layer addition and removal as extra feature.
  - *mixerFvMesh* – when sliding interface needed between one moving part and a fixed one.
  - *movingConeTopoFvMesh* – layers are added or removed depending on cell layer thickness.

Procedure to define a mesh with motion

- Firstly, the mesh is defined in the *blockMeshDict* inside the *constant* folder.

- In this same folder, the *dynamicMeshDict* is added where there is specified:
  - The chosen class.
  - The solver used, if needed.
  - The diffusivity model used, if needed.

dynamicInkJetFvMesh class (1/4)

```
#include "dynamicInkJetFvMesh.H"
#include "addToRunTimeSelectionTable.H"
#include "volFields.H"
#include "mathematicalConstants.H"
// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * *
* //
namespace Foam
{
    defineTypeNameAndDebug(dynamicInkJetFvMesh, 0);       //It call the functions
typeName and debug to specify the type class used, which is dynamicInkJetFvMesh in this case,
and some information for debugging.
    addToRunTimeSelectionTable(dynamicFvMesh, dynamicInkJetFvMesh,
IOobject);        //It adds the dynamicInkJetFvMesh (which is thisType, dynamicInkJetFvMesh,
inside the baseType, dynamicFvMesh) to the table where the classes used are defined
}
// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * *
* //
Foam::dynamicInkJetFvMesh::dynamicInkJetFvMesh(const IOobject& io)
:
    dynamicFvMesh(io),
    dynamicMeshCoeffs_
```

dynamicInkJetFvMesh class (2/4)

```
    (
        IOdictionary
        (
            IOobject
            (
                "dynamicMeshDict",
                io.time().constant(),          //dynamicMeshDict is located in the folder constant
                *this,
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            )
        ).subDict(typeName + "Coeffs")         //A subdictionary called dynamicInkJetFvMeshCoeffs exist inside the dynamicFvMesh with the following scalar numbers
    ),
    amplitude_(readScalar(dynamicMeshCoeffs_.lookup("amplitude"))),
    frequency_(readScalar(dynamicMeshCoeffs_.lookup("frequency"))),
    refPlaneX_(readScalar(dynamicMeshCoeffs_.lookup("refPlaneX"))),
    stationaryPoints_
    (
        IOobject
        (
            "points",
            io.time().constant(),              //the file points is also located in the folder constant
            meshSubDir,
            *this,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    )
{
    Info<< "Performing a dynamic mesh calculation: " << endl
```

dynamicInkJetFvMesh class (3/4)

```
        << "amplitude: " << amplitude_
        << " frequency: " << frequency_
        << " refPlaneX: " << refPlaneX_ << endl;
}
// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //
Foam::dynamicInkJetFvMesh::~dynamicInkJetFvMesh()
{}
// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //
bool Foam::dynamicInkJetFvMesh::update()          //member function for this class where the motion equation is defined and it updates the mesh
{
    scalar scalingFunction =
        0.5*(::cos(2*mathematicalConstant::pi*frequency_*time().value()) - 1.0);
    Info<< "Mesh scaling. Time = " << time().value() << " scaling: "
        << scalingFunction << endl;

    pointField newPoints = stationaryPoints_;    //new points are given the values of the stationary ones
    newPoints.replace
    (
        vector::X,
        stationaryPoints_.component(vector::X)*
        (
            1.0
          + pos
```

dynamicInkJetFvMesh class (4/4)

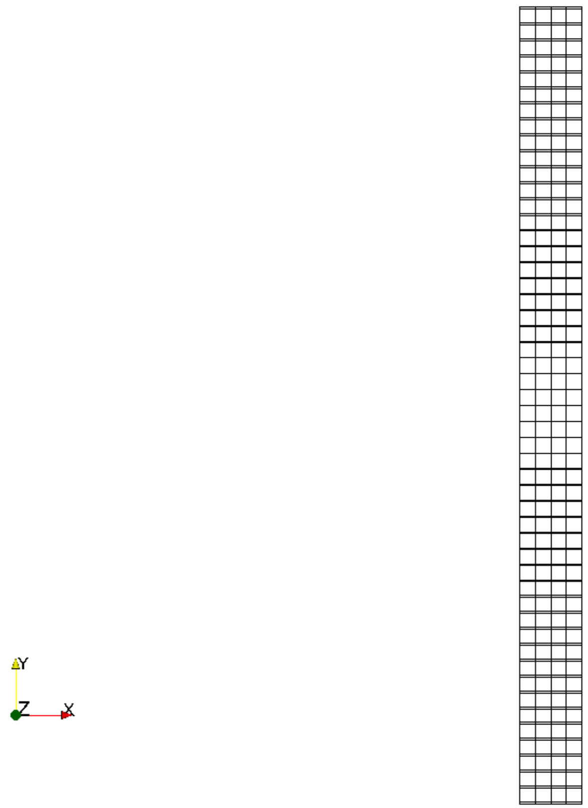
```
                (
                  - (stationaryPoints_.component(vector::X))
                  - refPlaneX_
                )*amplitude_*scalingFunction
            )
        );      //with the function replace the new points are recalculated following the motion
equation described just above. With vector::X specification it is said that the motion is only
changing the mesh in one direction, in this case in the X direction

    fvMesh::movePoints(newPoints);        //Mesh points are moved to the new points
calculated
    volVectorField& U =
        const_cast<volVectorField&>(lookupObject<volVectorField>("U"));
    U.correctBoundaryConditions();
    return true;
}
```

dynamicInkJetFvMesh example (1/8)

- Download the files from the website (Mesh Motion Alternatives by Andreu Oliver).
- Put it in your user directory.
- The example is done as follows:
  1. Create the example folder.
  2. Inside this folder put the next folders:
     - *0* (*p*, *U*).
     - *constant* (*dynamicMeshDict*, *polyMesh* and *transport properties*).
     - *system* (*controlDict*, *fvSchemes* and *fvSolution*)
  3. Define the mesh in the *blockMeshDict* file.

dynamicInkJetFvMesh example (2/8)



dynamicInkJetFvMesh example (3/8)

4. *dynamicMeshDict* is created, its code is:

```
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      motionProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dynamicFvMesh dynamicInkJetFvMesh;

motionSolverLibs ("libfvMotionSolvers.so");

dynamicInkJetFvMeshCoeffs
{
        amplitude       0.06;
        frequency       2;
        refPlaneX       0;
}

// ************************************************************************* //
```

dynamicInkJetFvMesh example (4/8)

- Simplification of the *icoDyMFoam* solver to the *icoDyMFoamMesh* one:

1. Copy the *icoDyMFoam* solver to your user directory and rename it:

```
>> cp -r $FOAM_SOLVERS/incompressible/icoDyMFoam \
$WM_PROJECT_USER_DIR/icoDyMFoamMesh
>> cd icoDyMFoamMesh
>> wclean
>> mv icoDyMFoam.C icoDyMFoamMesh.C
```

2. Make the modifications, by deleting:
    1. Make the fluxes absolute.
    2. Make the fluxes relative to the mesh motion.
    3. Pimple loop.

dynamicInkJetFvMesh example (5/8)

3. Before compiling the solver, *Make/files* should be:

```
icoDyMFoamMesh.C
EXE = $(FOAM_USER_APPBIN)/icoDyMFoamMesh
```

4. Now, the solver can be compiled:

```
>> wmake
```

- Finally, the new solver can be called and the results can be seen in *paraFoam*:

```
>> icoDyMFoamMesh
>> paraFoam
```

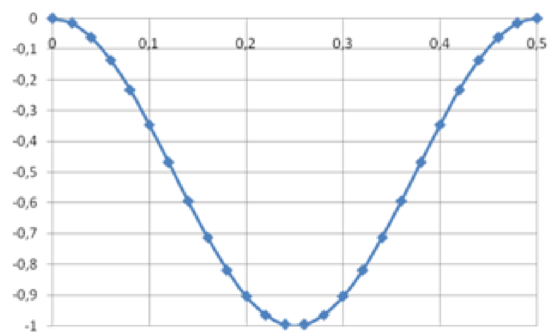
dynamicInkJetFvMesh example (6/8)

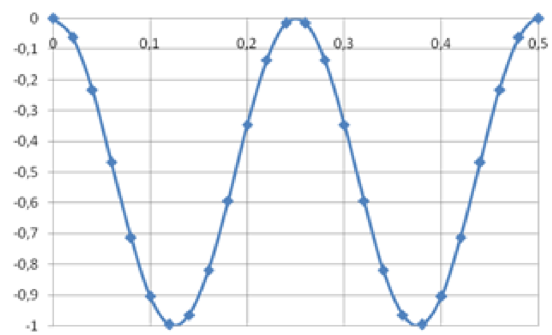
- The equation of motion is:

$$\text{scaling\_function} = 0.5 \cdot (\cos(2\pi t f) - 1) \qquad \text{Eq. 1}$$

$$x = x_{old} \cdot (1 + \text{pos}(-x_{old} - \text{refPlaneX}) \cdot \text{amplitude} \cdot \text{scaling\_function}) \qquad \text{Eq. 2}$$

- Effects of:
  - *amplitude*: varies the length that the mesh is deformed in x direction.
  - *frequency*: varies the speed of change.





dynamicInkJetFvMesh example (7/8)

- *refPlaneX*: it changes the reference plane. But depending on the interval the mesh is different:

  - *refPlaneX* $\in [-\infty, 0]$

    → Mesh motion for this interval of values.

  - *refPlaneX* $\in (0, 0.006]$

    → The points that have an x-position smaller than refPlaneX are moved, the rest are kept in the initial position.

  - *refPlaneX* $\in (0.006, +\infty]$

    → There is no motion.

dynamicInkJetFvMesh example (8/8)

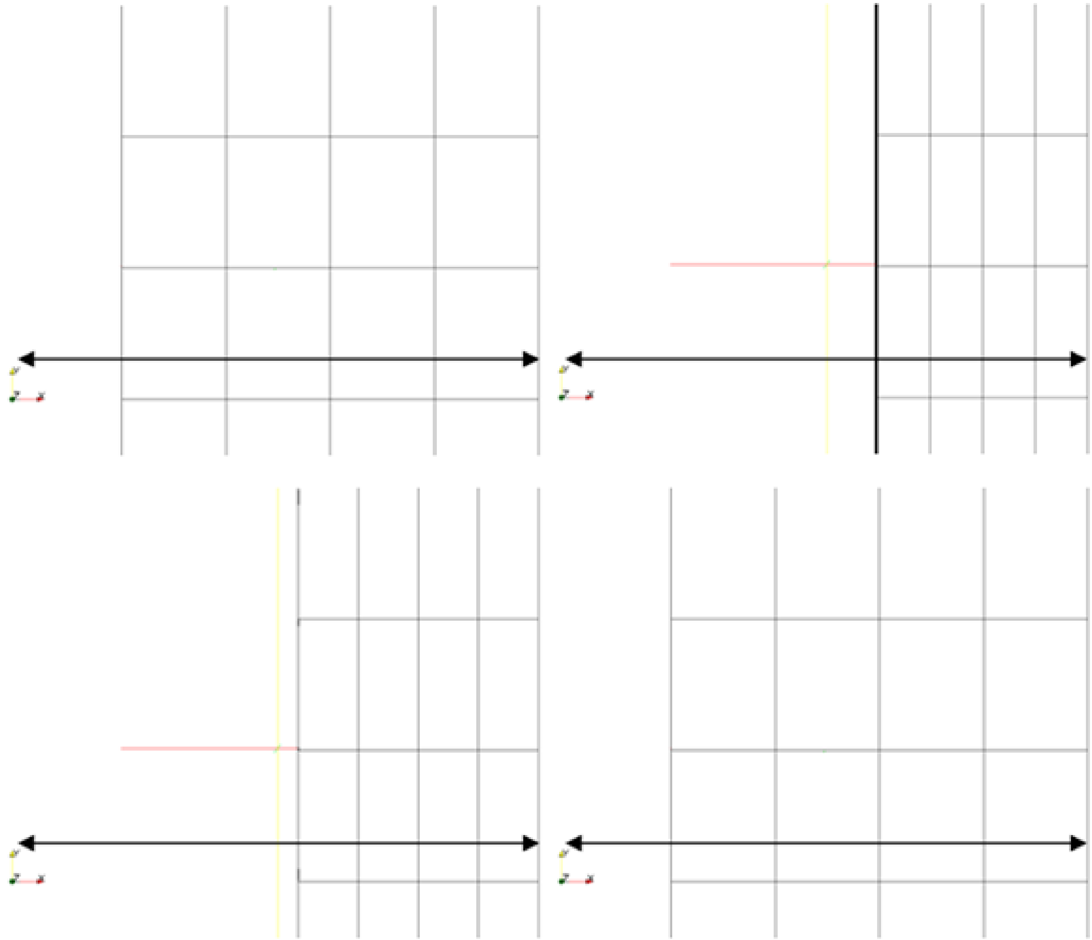
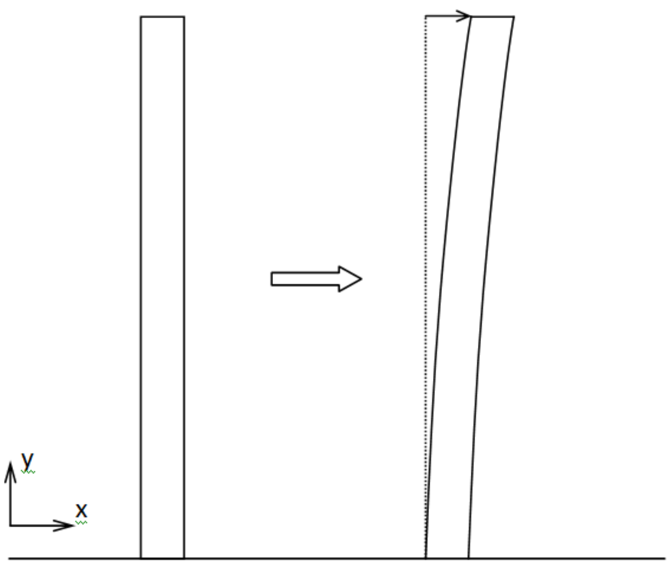


Figure 3: Mesh motion with 0.06m of amplitude, 2Hz of frequency and 0m of refPlaneX for t = 0.1, 0.25, 0.3 and 0.4s.

dynamicMyClassFvMesh class (1/4)

- It has a polynomial motion equation varied by a scaling function.



$$\text{scaling\_function} = \cos(2\pi t f) \qquad \text{Eq. 5}$$

$$x = x_{old} + a \cdot t \cdot y^2 \cdot \text{scaling\_function} \qquad \text{Eq. 6}$$

dynamicMyClassFvMesh class (2/4)

- The new class is created from the existing one:
  1. Copy and rename everything with the new class name

```
>> cp -r $FOAM_SRC/dynamicFvMesh/dynamicInkJetFvMesh \
$WM_PROJECT_USER_DIR/dynamicMyClassFvMesh
>> cd $WM_PROJECT_USER_DIR/dynamicMyClassFvMesh
>> sed s/dynamicInkJetFvMesh/dynamicMyClassFvMesh/g <dynamicInkJetFvMesh.C \
>dynamicMyClassFvMesh.C
>> sed s/dynamicInkJetFvMesh/dynamicMyClassFvMesh/g <dynamicInkJetFvMesh.H \
>dynamicMyClassFvMesh.H
>> rm -r dynamicInkJetFvMesh.*
>> cp -r $FOAM_SRC/dynamicFvMesh/Make $WM_PROJECT_USER_DIR/dynamicMyClassFvMesh
```

dynamicMyClassFvMesh class (3/4)

2. Modify the class (its equation and constants):

```
    a_(readScalar(dynamicMeshCoeffs_.lookup("a"))),
    frequency_(readScalar(dynamicMeshCoeffs_.lookup("frequency"))),
    // refPlaneX_(readScalar(dynamicMeshCoeffs_.lookup("refPlaneX"))),

        << "a: " << a_
        << " frequency: " << frequency_ << endl;
     //   << " refPlaneX: " << refPlaneX_ << endl;

    scalar scalingFunction =
        (::cos(2*mathematicalConstant::pi*frequency_*time().value());

    newPoints.replace
    (
        vector::X,
        stationaryPoints_.component(vector::X)+
        a-*time().value()*(stationaryPoints_.component(vector::Y))*
(stationaryPoints_.component(vector::Y))*scalingFunction
    );
```

dynamicMyClassFvMesh class (4/4)

3. To compile the *dynamicMyClassFvMesh*:

- *files* should be changed to:

```
dynamicMyClassFvMesh.C
LIB=$(FOAM_USER_LIBBIN)/libdynamicMyClassFvMesh
```

- The next line should be added to *options* file:

```
-I$(LIB_SRC)/dynamicFvMesh/lnInclude
```

- Compilation of the class can be done:

```
>> cd $WM_PROJECT_USER_DIR/dynamicMyClassFvMesh
>> wmake libso
```

dynamicMyClassFvMesh example (1/3)

- Copy the content of myExample to myClassExample:

```
>> cp -r $WM_PROJECT_USER_DIR/myExample $WM_PROJECT_USER_DIR/myClassExample
```

- Some changes have to be done in dynamicMeshDict:

```
dynamicFvMeshLibs       ("libdynamicMyClassFvMesh.so");
dynamicFvMesh           dynamicMyClassFvMesh;
motionSolverLibs        ("libfvMotionSolvers.so");
dynamicMyClassFvMeshCoeffs
{
        a       0.4;
        frequency       2;
}

// ************************************************************************* //
```

- Now the mesh can be moved typing:

```
>> icoDyMFoamMesh
>> paraFoam
```

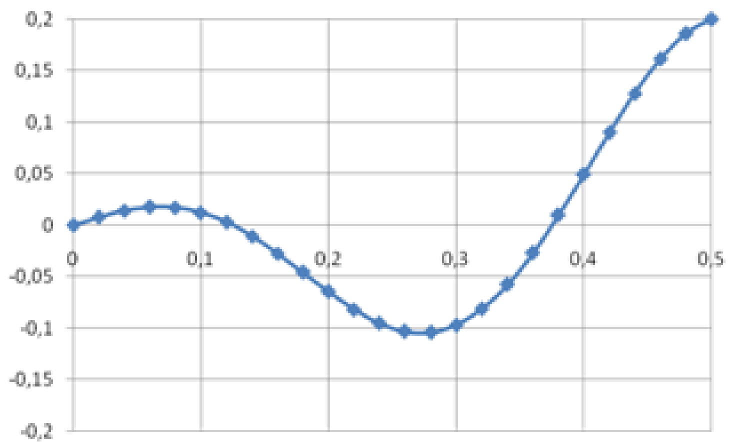
dynamicMyClassFvMesh example (2/3)

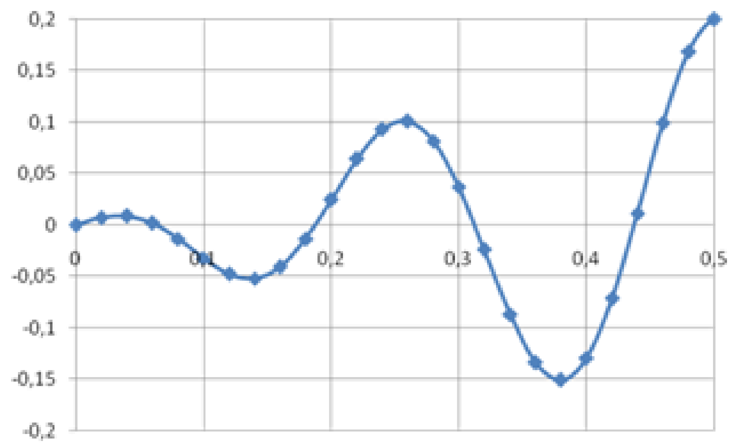
- The equation of motion is:

$$\text{scaling\_function} = \cos(2\pi t f) \qquad \text{Eq. 5}$$

$$x = x_{old} + a \cdot t \cdot y^2 \cdot \text{scaling\_function} \qquad \text{Eq. 6}$$

- The effects of:
  - *a*: varies the total displacement in x direction.
  - *frequency*: varies the speed of change.





dynamicMyClassFvMesh example (3/3)

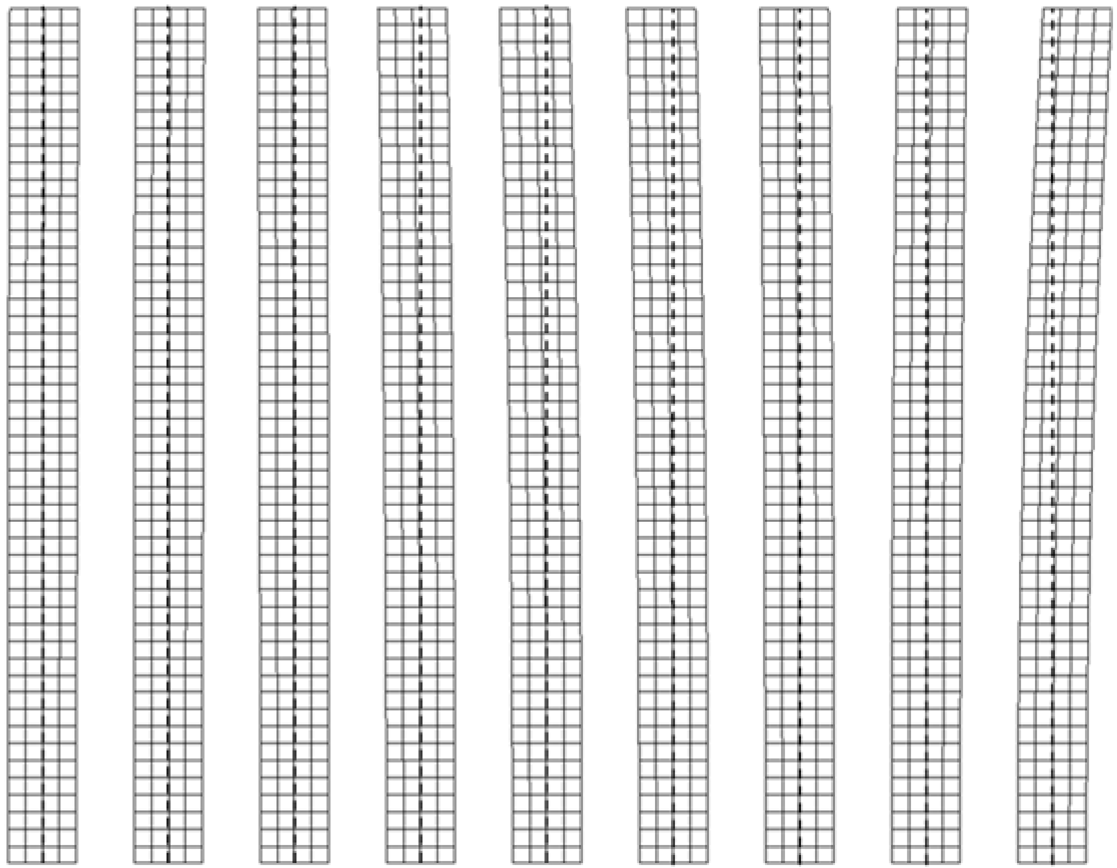


Figure 12: Mesh motion with 0.8m of amplitude and 2Hz of frequency for t = 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4 and 0.5s.

Conclusions

- There are two approaches for mesh manipulation:
  - Automatic mesh motion.
  - Topological changes in the mesh.
- There are different classes for each one.
- dynamicInkJetFvMesh defines a movement based on harmonic motion.
- dynamicMyClassFvMesh defines a sinusoidal motion around y direction depending on y position.

THANK YOU FOR LISTENING!

Any questions?